

TITLE OF THE INVENTION

**METHOD AND SYSTEM FOR OBTAINING DEVICE SERVICES ON A
SELF-SERVICE FINANCIAL TRANSACTION TERMINAL**

5

PRIORITY APPLICATION

- This application claims priority to co-pending US Provisional Application No. 60/162,994, dated November 1, 1999, entitled "Method And System For Extensions For Financial Services Service Provider Framework For A Self-Service Transaction Terminal (XFS Service Provider Framework)," and is incorporated herein by reference.

CROSS REFERENCE TO RELATED APPLICATIONS

- This application relates to Attorney Docket No. CITI0198/T0091-195417, filed simultaneously, entitled "Method And System For Secure Communication Between A Self-Service Financial Transaction Terminal And A Remote Operator Interface," and is incorporated herein by reference.

- This application relates to Attorney Docket No. CITI0200/T0091-195419, filed simultaneously, entitled "Method And System For Installing And/Or Upgrading Software On A Self-Service Financial Transaction Terminal From A Remote Computer," and is incorporated herein by reference.

- This application relates to Attorney Docket No. CITI0202/T0091-195420, filed simultaneously, entitled "Method And System For Simultaneous And Unattended Installation Of Software On A Self-Service Financial Transaction Terminal," and is incorporated herein by reference.

This application relates to Attorney Docket No. CITI0199/T0091-195421, filed simultaneously, entitled "Method And System For Remote Operator Interface With A Self-Service Financial Transaction Terminal," and is incorporated herein by reference.

This application relates to Attorney Docket No. CITI0201/T0091-195422, filed simultaneously, entitled "Method And System For Coordinating Session Activities At A Self-Service Financial Transaction Terminal," and is incorporated herein by reference.

- 5 This application relates to Attorney Docket No. CITI0203/T0091-195578, filed simultaneously, entitled "Method And System For Configuration Of Self-Service Financial Transaction Terminals For A Common Software Release," and is incorporated herein by reference.

- 10 This application relates to US Provisional Application No. 60/162,673, filed November 1, 1999, entitled "Method And System For Secure Communication Between A Self-Service Transaction Terminal And A Remote Operator Interface (Remote Operator Interface Security)," and is incorporated herein by reference.

- 15 This application relates to US Provisional Application No. 60/163,002, filed November 1, 1999, entitled "Method And System For Installing And/Or Upgrading Software On A Self-Service Financial Transaction Terminal From A Remote Computer (Remote Installation/Software Upgrade)," and is incorporated herein by reference.

- 20 This application relates to US Provisional Application No. 60/162,815, filed November 1, 1999, entitled "Method And System For Simultaneous And Unattended Installation Of Software On A Self-Service Financial Transaction Terminal (Global Installation Framework)," and is incorporated herein by reference.

- 25 This application relates to US Provisional Application No. 60/163,000, filed November 1, 1999, entitled "Method And System Of Remote Operator Interface For A Self-Service Financial Terminal (Remote Operator Interface)," and is incorporated herein by reference.

This application relates to US Provisional Application No. 60/162,816 filed November 1, 1999, entitled "Method And System For Coordinating Session Activities At A Self-Service Financial Transaction Terminal (ATM Session Manager)," and is incorporated herein by reference.

5 This application relates to US Provisional Application No. 60/162,672, filed November 1, 1999, entitled "Method And System For Configuration Of Self-Service Financial Terminals For A Common Software Release (Framework For Configuration Of Self-Service Financial Terminals)," and is incorporated herein by reference.

COPYRIGHT NOTIFICATION

10 A portion of the disclosure of this patent document and its figures contain material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the
15 patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyrights whatsoever.

BACKGROUND OF THE INVENTION

1. Field of the Invention

20 This invention relates generally to the field of self-service financial transaction terminals, such as automatic teller machines (ATMs), and more particularly to a method and system for obtaining ATM device services using a service provider framework.

2. Background

25 Typically, a financial institution, such as a bank, develops its own proprietary ATM hardware and software, in addition to ATM hardware and

software which it may purchase from a variety of ATM vendors. Implementation of systems from other makers can be difficult because, for example, of difficulty in interfacing and reusing present application resources. ATMs contain special devices, such as cash dispensers, card readers, and printers, which require a specialized interface for control from applications. Previously, hardware makers' interfaces were proprietary, and the Windows Open Services Architecture Extensions for Financial Services (WOSA/XFS) was developed to define a standard for application control of specialized ATM peripherals.

The term WOSA/XFS defines a standard for an interface with which applications, such as ATM applications, can get device services, such as making the cash dispenser dispense cash or reading a customer's transaction card. Using the WOSA/XFS standard, applications and back-end services are connected through the unified set of an Application Programming Interface (API) and a Service Provider Interface (SPI) in a Windows environment. Thus, applications only have to be conscious of the WOSA interface and not of the implementation of various services. Development by a financial institution of its own WOSA/XFS-compliant software can assure the portability of the financial institution's software to ATMs provided by a variety of vendors.

SUMMARY OF THE INVENTION

It is a feature and advantage of the present invention to provide a method and system for obtaining ATM device services that enables a financial institution to have its applications require device services on a variety of vendors' ATM machines, as well as its own ATM machines.

It is another feature and advantage of the present invention to provide a method and system for obtaining ATM device services that allows the financial institution's service provider developers to be concerned only with the code specific and the device specific code for a particular device.

It is an additional feature and advantage of the present invention to provide a method and system for obtaining ATM device services that enables a financial institution to develop service providers in-house in a timely fashion with the financial institution's source code and within the proper control of the financial institution.

To achieve the stated and other features, advantages and objects, an embodiment of the present invention makes use of XFS-compliant service providers to enable a financial institution, such as a bank, in the implementation of its ATM system, to have its applications require device services on a variety of vendors' ATM machines, as well as its own ATM machines, such that getting device services on a particular vendor's platform is the same as getting device services on its own platform. The method and system for an embodiment of the present invention provides all the common processing that, in the development of a service provider, is common to every service provider and allows the service provider developer to be concerned only with the code-specific and the device-specific code for a particular device. Thus, development of service providers can be provided in-house in a timely fashion with the financial institution's source code and within the proper control of the financial institution.

An embodiment of the present invention makes use of a service provider framework in which data is received by the ATM application, such as a customer making a selection on the ATM touchscreen, that indicates to the ATM application that there is a need for the performance of an ATM device function, such as a depository function, a printer function, a card reader function, a safe door function, a cash dispenser function, or a touchscreen function. The ATM application makes a sub-routine call, referred to as a WFS request, to a lower level layer of central ATM monitoring and management application software to request device service from a service provider. The sub-routine call is translated at the lower level layer into a function category request, referred to as a WFP request, by an XFS manager as an entry point into the service provider for

processing by the service provider. The selection of function category requests include, for example, a WFPCancelAsyncRequest request, a WFPCLose request, a WFPDeregister request, a WFPExecute request, a WFPGetInfo request, a WFPFLock request, a WFPOpen request, a WFPRegister request, a
5 WFPSetTraceLevel request, a WFPUnloadService request, and a WFPUnlock request. After processing the request, the service provider returns a result to the ATM application.

In an embodiment of the present invention, the ATM application receives the data indicative of a need for obtaining an ATM device service in connection
10 with an ATM device, such as a depository, a printer, a card reader, a safe door, a cash dispenser, and/or a touchscreen. The ATM application issues a request to the XFS manager to get the ATM device service by making a sub-routine call to the XFS manager to get the ATM device service from a service provider. The XFS manager translates the sub-routine call as an entry point into the service
15 provider for processing by the service provider, and a request object associated with the request is instantiated. The service provider is implemented by instantiating an instance of the service provider framework XFS service provider base class and one or more instances of the service provider framework request objects required to process the request.

The service provider for an embodiment of the present invention
20 instantiates a specific instance of the service provider's service provider request object which is derived from the XFS service provider base class service provider request object class hierarchy. The basic unit in which service provider processing is performed is the request object, and there is a request object defined
25 for eleven WFP requests, comprising a SpiRequest class, a SpiAsyncRequest class, a request specific class, such as OpenRequest or Execute Request, and optionally, a service provider specific request class, one of which is derived from the other. As WFP requests are made, the service provider framework invokes virtual methods within derived objects of the service provider through class

inheritance to allow the service provider to perform processing unique to the particular ATM device service.

The WFP requests for an embodiment of the present invention are processed in two parts, the first part being called the immediate processing part and the second part being called the deferred processing part. Immediate processing is performed in the same thread as used by the XFS manager when invoking an entry point of the service provider. The service provider framework performs parameter verification in its immediate processing method. If no errors are encountered, the service provider framework invokes the
5 spImmediateProcessing() method within the service provider's derived class. At this point, the service provider can perform more specific, although not complete, parameter verification. The return code from the spImmediateProcessing() method is returned to the XFS manager. If the service provider has not implemented a spImmediateProcessing method, the service provider framework
10 determines which code is returned to the XFS manager.

If the service provider for an embodiment of the present invention has not implemented the spDeferredProcessing() method, the service provider framework posts the request complete event. Exceptions to this include the WFPCancelAsyncRequest, WFPSetTraceLevel, and WFPUnload requests, which
15 are process immediate request and have no deferred processing methods. The process immediate requests are handled completely by the service provider framework and require no processing on the part of the service provider. The service provider framework uses, for example, four threads per service provider to manage WFP requests for a particular service provider, including a deferred
20 processing queue manager thread and a deferred processing thread. WFP requests are placed on a deferred processing queue, and when signaled, the deferred processing queue manager thread pops a request object from the deferred processing queue and creates a deferred processing thread in which the deferred
25 processing for the particular WFP request is performed.

In an embodiment of the present invention, when a WFP request is dequeued, a processing thread is created and the service provider framework performs the deferred processing for the particular request that is common to all service providers. Upon successful completion of that processing, the service provider framework invokes a `spDeferredProcessing()` method within a derived class of the service provider, and the service provider performs all processing necessary to satisfy the request within the `spDeferredProcessing` method, including posting a request complete event. When the deferred processing method returns, the thread is terminated, and the request object is deleted. If the service provider has not implemented the `spDeferredProcessing()` method, the service provider framework posts a request complete.

In an embodiment of the present invention, the service provider accesses one or more request parameters for one or more WPF requests. WPF requests include, for example, a `WFPCancelAsyncRequest` request, a `WFPCLose` request, a `WFPDeregister` request, a `WFPEXecute` request, a `WFPGetInfo` request, a `WFPLOCK` request, a `WFPOpen` request, a `WFPRegister` request, a `WFPSetTraceLevel` request, a `WFPUnloadService` request, and a `WFPUnlock` request. The request parameters for the `WFPCancelAsyncRequest` request include an `hService` parameter and a `reqID` parameter; the request parameters for the `WFPCLose` request include an `hService` parameter, an `hWnd` parameter, and a `reqID` parameter; the request parameters for the `WFPDeregister` request include an `hService` parameter, a `dwEventClass` parameter, an `hWndReg` parameter, an `hWnd` parameter, and a `reqID` parameter; and the request parameters for the `WFPEXecute` request include an `hService` parameter, a `dwCommandData` parameter, an `IpCommandData` parameter, a `dwTimeOut` parameter, an `hWnd` parameter, and a `reqID` parameter.

In addition, the request parameters for the `WFPGetInfo` request include an `hService` parameter, a `dwCategory` parameter, an `IPQueryDetails` parameter, a `dwTimeOut` parameter, an `hWnd` parameter, and a `reqID` parameter; the request

parameters for the WFPOpen request include an hService parameter , an
IpszLogicalName parameter , an hApp parameter , an IpszApplicationID
parameter, a dwTraceLevel parameter , a dwTimeOut parameter , an hWnd
parameter , a reqID parameter , an hPprovider parameter , a
5 dwSPVersionsRequired parameter , an IpSPIVersion parameter , a
dwSrcvVersionsRequired parameter , and an IpSrcvVersion parameter; the
request parameters for the WFPRegister request include an hService parameter, a
dwEventClass parameter, an hWndReg parameter, an hWnd parameter, and a
reqID parameter; and the request parameters for the WFPSetTraceLevel request
10 include an hService parameter and a dwTraceLevel parameter.

Additional objects, advantages, and novel features of the invention will be
set forth in part in the description which follows, and in part will become more
apparent to those skilled in the art upon examination of the following, or may be
learned by practice of the invention.

BRIEF DESCRIPTION OF THE FIGURES

Fig. 1 is a flow chart which illustrates an example of the process of
requesting device services, such as dispensing cash, for an embodiment of the
present invention;

Fig. 2 is a diagram which illustrates an example of XFS service provider
hierarchy, using a depositor example, for an embodiment of the present invention;

Fig. 3 is a diagram which illustrates an example of service provider
request object class hierarchy, using a WFPEXecute request example, for an
embodiment of the present invention;

Fig. 4 is a table that shows examples of WFPCancelAsyncRequest
parameters and how each accessed for an embodiment of the present invention;

Fig. 5 is a table that shows examples of WFPCLose parameters and how
each is accessed for an embodiment of the present invention.

Fig. 6 is a table which illustrates examples of WFPDeregister parameters and how each is accessed for an embodiment of the present invention;

Fig. 7 is a table which illustrates examples of WFPEXecute parameters and how each is accessed for an embodiment of the present invention;

5 Fig. 8 is a table that shows examples of WFPGetInfo parameters and how each is accessed for an embodiment of the present invention;

Fig. 9 is a table which illustrates examples of WFPOpen parameters and how each is accessed for an embodiment of the present invention;

10 Fig. 10 is a table which illustrates examples of WFPRegister parameters and how each is accessed for an embodiment of the present invention; and

Fig. 11 is a table which illustrates examples of WFPSetTraceLevel parameters and how each is accessed for an embodiment of the present invention.

DETAILED DESCRIPTION

15 Referring now in detail to an embodiment of the present invention, an example of which is illustrated in the accompanying attachment, the method and system for an embodiment of the present invention makes use of XFS-compliant service providers to enable a financial institution, such as a bank, in implementation of its ATM system, sometimes referred to herein as global ATM, to have its applications require device services on a variety of vendors' ATMs, as well as its own ATMs. In that the way, getting device services on a particular vendor's platform is the same as getting device services on the financial institution's own platform. The XFS service provider framework for an embodiment of the present invention provides all the common processing that, in the development of a service provider, is common to every service provider. 25 There is one service provider per device, and an embodiment of the present invention provides the common processing that each service provider is required to perform. This allows the service provider developer to be concerned only about the code-specific and the device-specific code for a particular device.

The software for an embodiment of the present invention is meant to run on the financial institution's ATMs. The other vendors provide their own service providers for their own devices. It is at the device level that the differences appear, which become quite pronounced between different ATM vendors.

5 Typically, applications which are visible to the customer can run readily from one vendor to another. However, getting device services is typically quite different from one vendor to the other. The service providers provide a layer to make that invisible from the applications. In an embodiment of the present invention, the XFS interface is presented to the applications, and the applications are not
10 concerned with what goes on below that. An aspect of embodiment of the present invention includes development of these service providers. The framework for an embodiment of the present invention makes use of a number of files, referred to as "H" files and a Dynamic Link Library (DLL) from which is compiled and to which is linked in order to develop what is called a service provider. The method
15 and system of the present invention is used only in conjunction with the applications which make device requests for device services. Terms, such as "queuing" and "WFS request," as used herein are defined in the XFS standard, with which the framework for an embodiment of the present invention is compliant. For example, the term "WFS" is a prefix indicating a particular
20 function category.

Fig. 1 is a flow chart which illustrates an example of the process of requesting device services, such as dispensing cash, for an embodiment of the present invention. Referring to Fig. 1, at S1, in connection with dispensing cash at the ATM, the application, which is a thing the customer sees, decides in the
25 course of its processing that it needs to dispense cash. There is an XFS standard defined for dispensing cash, and the application knows that at that point it can issue the proper command, which is called a WFS execute command. At S2, the WFS execute command is translated by the XFS manager to a WFP execute command for processing by the appropriate (in this case cash dispenser) service

provider. The term "WFP" is another prefix indicating a particular function category, and the WFP execute command goes to the appropriate service provider. At S3, the service provider drives the cash dispenser and performs the appropriate low level hardware commands to make the cash dispenser present and distribute cash. The service provider returns the result to the XFS manager, and at S4, the XFS manager returns the result back to the application. A difference between WFS commands and WFP commands, for example, is that WFS commands are made by ATM applications to get device services from service providers, and WFP commands are the function entry points "into" the service provider that the framework for an embodiment of the present invention "exposes" to the XFS manager. The XFS manager translates WFS commands into the corresponding WFP command.

The method and system for an embodiment of the present invention is used in customer sessions other than dispensing cash. Other functions include, but are not limited to, for example, depositing, printing, managing the ATM card reader, as well as the function of managing and keeping track of when the safe door opens or closes and taking the ATM out of service when that occurs. There are basically five devices that are involved directly with customer sessions, such as dispensing cash, depositing an envelope, printing a receipt or ticket, driving the touchscreen, and also the card reader. In the method and system for an embodiment of the present invention, there is a button on the ATM touchscreen, and if the button pressed, a certain amount of cash is be dispensed to the customer. When the customer touches the touchscreen it appears, for example, as a mouse event to the ATM application. In other words, it is like someone clicking with a mouse. When an application gets notification that the button has been touched, it makes an Object Linking and Embedding (OLE) call, which is basically a sub-routine call, to a lower level item of software, and that sub-routine call is then translated at the lower layer of software into a WFP execute command. That is the entry point into the framework for an embodiment of the

present invention. At that point, the framework passes an entry point that is below the framework into the specific service provider, and the service provider then has control and can issue the proper device commands to get the cash.

An embodiment of the present invention provides a solution for a financial institution because, for example, it can be provided in-house in a timely fashion with the financial institution's source code and within the proper control of the financial institution. Turning now to a description of the class structure and basic processing of the XFS service provider framework for an embodiment of the present invention, the framework provides the service provider developer with a set of C++ (object-oriented programming language) objects that perform the necessary processing to be an XFS compliant service provider. The developer then needs only to provide that code unique to the specific service provider.

In an embodiment of the present invention, a service provider is implemented by instantiating an instance of the service provider framework XFS service provider base class and instances of the framework request objects. As WFP requests are made, the service provider framework through C++ class inheritance, invokes virtual methods within the service provider's derived request objects to allow the service provider to perform processing unique to that service. Fig. 2 is a diagram which illustrates an example of XFS service provider hierarchy, using a depositor example, for an embodiment of the present invention. The XFS service provider base class 10 contains the service provider framework objects required to process WFP requests. Each service provider is required to instantiate a specific instance 12 of it's service provider object. That instance 12 must be derived from the XFS service provider base class 10 as shown in Fig. 2.

Fig. 3 is a diagram which illustrates an example of service provider request object class hierarchy, using a WFPExecute request example, for an embodiment of the present invention. The basic unit in which all service provider processing is performed, is the request object. There is a request object defined, for example, for eleven WFP requests. The request object comprises a

SpiRequest class 14, a SpiAsyncRequest class 16, a request specific class, such as OpenRequest or ExecuteRequest 18, and optionally, a service provider specific request class 20. One is derived from the other as shown in Fig. 3, which uses the WFPExecute request as an example.

- 5 In an embodiment of the present invention, each time the XFS manager invokes one of a service provider's WFP entry points, a request object associated with that request is instantiated. WFP requests are processed in two parts. The first part is referred to as "immediate processing", and the second part is referred to as "deferred processing". Immediate processing is performed in the same
- 10 Windows thread as that used by the XFS manager when invoking a service provider's WFP entry point. The service provider framework performs parameter verification in its immediate processing method. If no errors are encountered, the service provider framework invokes the spImmediateProcessing() method within the service provider's derived class. At this point, the service provider can
- 15 perform more specific, although not complete, parameter verification. The return code from spImmediateProcessing() is returned to the XFS manager. If the service provider has not implemented a spImmediateProcessing method, the service provider framework determines which code is returned to the XFS manager.
- 20 Once the immediate processing has completed without errors, in an embodiment of the present invention, a request timer for that request is enabled and the request object is placed on the deferred processing queue. This ensures proper serialization of request processing as required by the WOSA standard. When dequeued, a processing thread is created and the service provider
- 25 framework performs the deferred processing for that request that is common to all service providers. If that completes successfully, the service provider framework invokes the spDeferredProcessing() method within the service provider's derived class. Within this method, the service provider performs all the processing necessary to satisfy the request including posting the request complete event.

When the deferred processing method returns, the thread is terminated and the request object is then deleted. If the service provider has not implemented the `spDeferredProcessing()` method, the service provider framework posts the request complete event.

5 An exception to the foregoing are the `WFPCancelAsyncRequest`, `WFPSetTraceLevel`, and the `WFPUnload` requests. These are “process immediate” requests, and as such, they have no deferred processing methods. All of these requests are completely handled by the service provider framework and require no processing on the part of the service provider.

10 A WFP request can be interrupted, for example, if the request timer expires or if the request is cancelled. If either of those events occur, a method within the request object is called, depending on the state of the request. If the request is currently processing, the abort method in the service provider’s request object is called. If the request is queued, then the `abortQueued` method is called.

15 Both the abort and `abortQueued` methods are called with a parameter that indicates the reason for the interruption. Its values can be either `WFS_ERR_TIMEOUT` or `WFS_ERROR_CANCELED`. Both abort methods give the service provider the opportunity to properly clean up the request.

20 If not overridden, the service provider framework sends request complete messages for both the abort and `abortQueued` methods. That is all the processing the service provider framework performs for these operations. It is the responsibility of the service provider to implement either of these methods if the request complete messages sent by the service provider framework are undesirable.

25 The service provider framework for an embodiment of the present invention uses, for example, four threads per service provider to manage WFP requests for a particular service provider. The thread in which the XFS Manager uses to invoke the WFP entry point is not included in that number. The threads

include, for example, a deferred processing queue (DPQ) manager thread and a deferred processing thread.

As previously mentioned, in an embodiment of the present invention, WFP requests are placed on a deferred processing queue to ensure proper serialization of WFP requests. At initialization, the DPQ manager thread is created and is initially blocked. In other word, it is in a nonsignaled state awaiting a request object to be queued. When signaled, the DPQ manager thread pops a request object from the deferred processing queue and creates another thread called the deferred processing thread in which the actual deferred processing for that WFP request is performed. The DPQ manager thread is again blocked while it awaits the termination of the deferred processing thread. When unblocked, the DPQ manager thread loops back and processes all request objects that may have been queued. If none are queued, then the DPQ manager thread blocks awaiting the next WFP request.

The main processing for most WFP requests is performed in its deferred processing. As already mentioned, a deferred processing thread is created each time a request object is popped from the deferred processing queue. Since a request object is not popped from the DPQ until the thread from the previous request object has terminated, there can be only one deferred processing thread active at any given time. The service provider may create other threads during a request object's deferred processing. If so, then the service provider is responsible for managing that thread and determining when it should be terminated.

Turning now to a description, of how the service provider accesses the request parameters for each WFP request, if any, and the processing responsibilities of both the service provider, if any, and the service provider framework, WFP requests include, for example, WFPCancelAsyncRequest, WFPClose, WFPDeregister, WFPExecute, WFPGetInfo, WPFLock, WFPOpen, WFPRegister, WFPSetTraceLevel, WFPUnloadService, and WFPUnlock.

In an embodiment of the present invention, the WFPCancelAsyncRequest specifies to the service provider framework which request(s) to cancel. The request can be, for example, in either a queued state or a currently processing state. In both cases, the abort method in the request object is called by the service provider framework to allow the service provider to post the request complete notification. If the service provider has not implemented an abort method for that request, the service provider framework posts the request complete notification. The parameters to the WFPCancelAsyncRequest request are stored within the Cancel Request object. Those stored in the SpiRequest objects are private and can only be accessed using an accessing function. Fig. 4 is a table that shows an example of each WFPCancelAsyncRequest parameter and how it is accessed for an embodiment of the present invention.

With regard to the WFPClose request, the service provider framework maintains a table of active sessions for every service provider. Upon receiving a valid close request, the service provider framework removes the particular session from the table and de-registers all events associated with that session. The service provider may use the getNumberOfSessions() method within the Close Request object to obtain the number of sessions for the particular service provider. If the number is zero, the service provider is responsible for de-activating and unloading the PDH associated with that service provider.

The parameters to the WFPClose request are stored within the Close Request object. Those stored in the SpiRequest and SpiAsyncRequest objects are private and can only be accessed using a function. Fig. 5 is a table of each WFPClose parameter and how it is accessed. The method for immediate processing is:

HRESULT <specificProvider>CloseRequest::spImmediateProcessing(void).

The service provider framework verifies each of the WFPClose parameters. The service provider may find it unnecessary to implement this method. The method for deferred processing is:

Boolean <specificProvider>CloseRequest::spDeferredProcessing(void).

If the number of active sessions is zero, then spDeferredProcessing should deactivate and unload the PDH dynamic link library (dll) associated with the particular service provider.

- 5 The getNumberOfSessions method returns the number of currently active sessions. The method is:

```
int CloseRequest::getNumberOfSessions( serviceProvider )  
XfsServiceProvider * serviceProvider.
```

- 10 The parameters include serviceProvider which is a handle unique to each XFS service provider. This handle is provided to each service provider in the serviceProvider attribute in every Open Request object. For the return value, if serviceProvider is valid, the number of active sessions is returned. Otherwise -1 is returned.

- 15 The number of active sessions reflects the session referred to in the current close request object. Thus, if a service provider receives a WFPOpen and subsequently a WFPClose request, getNumberOfSessions will return zero if called from the spDeferredProcessing method within the close request. This assumes that the WFPOpen and WFPClose refer to the same session and that no other WFPOpens have been received for other sessions.

- 20 The service provider framework handles all processing associated with the WFPDeregister request. The parameters to the WFPDeregister request are stored within the Deregister Request object. Those stored in the SpiRequest and SpiAsyncRequest objects are private and can only be accessed using a function. Fig. 6 is a table which illustrates examples of each WFPDeregister parameter and
25 how it is accessed for an embodiment of the present invention.

 The parameters to the WFPEXecute request are stored within the Execute Request object. Those stored in the SpiRequest and SpiAsyncRequest objects are private and can only be accessed using a function. Fig. 7 is a table which

illustrates an example of each WFPEXecute parameter and how it is accessed for an embodiment of the present invention.

For immediate processing, the method is:

HRESULT <specificSp>ExecuteRequest:: spImmediateProcessing(void).

- 5 This method, if implemented, gives the service provider the opportunity to perform parameter verification before the Execute Request object is queued by the framework. The service provider framework verifies all parameters except dwCommand and lpCommandData. If any of those are invalid, this method is not invoked and the appropriate error code is returned to the application. The service
- 10 provider should verify that dwCommand is valid and supported. If not, WFS_ERR_INVALID_COMMAND or WFS_ERR_UNSUPP_COMMAND should be returned as appropriate. Because WFS_INVALID_DATA can only be returned in the Execute Complete event (not as an immediate error), the actual data pointed to by lpCommandData cannot be verified in this method. However
- 15 the service provider should verify that lpCommandData and any other pointers that it may contain are valid. If any are invalid, this method should return WFS_INVALID_POINTER. If the service provider returns WFS_SUCCESS, the request object is queued on the deferred processing queue for later processing.

For deferred processing, the method is:

- 20 Boolean <specificSp>ExecuteRequest:: spDeferredProcessing(void). This method, if implemented, is the entry point at which all WFPEXecute commands for this service provider are processed. The WFPEXecute parameters are accessed the same way as in the spImmediateProcessing method described above. Within this method the data contained in lpCommandData is verified.
- 25 The service provider is responsible for using XfsEvent::postEvent() for all events associated with the WFPEXecute request. This includes the Execute Complete event.

For abort processing, the method is:

void <specificSp>ExecuteRequest::abort(HRESULT abortReason, int
requestStatus).

This method, if implemented, can be invoked either because the request was
canceled by WFSCancel (abortReason = WFS_ERR_CANCELED) or because
5 the request timed out (abortReason = WFS_ERR_TIMEOUT). In either case, the
service provider is responsible for terminating any threads that may have been
created in the spDeferredProcessing method (see above) and posting the Execute
Complete event.

The service provider framework processes the WFPGetInfo request
10 differently from all other WFP requests. Specifically, the WFPGetInfo is placed
in a separate deferred processing queue, which means the deferred processing for
this request can never be blocked by some other non-WFPGetInfo request.

The parameters to the WFPGetInfo request are stored within the GetInfo
Request object. Those stored in the SpiRequest and SpiAsyncRequest objects are
15 private and can only be accessed using a function. Fig. 8 is a table of each
WFPGetInfo parameter and how it is accessed for an embodiment of the present
invention.

For immediate processing, the method is:

HRESULT <specificSp>GetInfoRequest:: spImmediateProcessing(void)

20 This method, if implemented, gives the service provider the opportunity to
perform parameter verification before the GetInfo Request object is queued by the
framework. The service provider framework verifies all parameters except
dwCategory and lpQueryDetails. If any of those are invalid, this method is not
invoked and the appropriate error code is returned to the application. The service
25 provider should verify that dwCategory is valid and supported. If not,
WFS_ERR_INVALID_CATEGORY or WFS_ERR_UNSUPP_CATEGORY
should be returned as appropriate. Because WFS_INVALID_DATA can only be
returned in the GetInfo Complete event (not as an immediate error), the actual
data pointed to by lpQueryDetails cannot be verified in this method. However the

service provider should verify that lpQueryDetails and any other pointers that it may contain are valid. If any are invalid, this method should return WFS_INVALID_POINTER. If the service provider returns WFS_SUCCESS, the request object is queued on the deferred processing queue for later processing.

5 For deferred processing, the method is:

Boolean <specificSp>GetInfoRequest:: spDeferredProcessing(void)

This method, if implemented, is the entry point at which all WFPGetInfo commands for this service provider are processed. The WFPGetInfo parameters are accessed the same way as in the spImmediateProcessing method described
10 above. Within this method, the data contained in lpQueryDetails is verified. The service provider is responsible for using XfsEvent::postEvent() for all events associated with the WFPGetInfo request. This includes the GetInfo Complete event.

For abort processing, the method is:

15 void <specificSp>GetInfoRequest::abort(HRESULT abortReason, int requestStatus).

This method, if implemented, can be invoked for either because the request was canceled by WFSCancel (abortReason = WFS_ERR_CANCELED) or because the request timed out (abortReason = WFS_ERR_TIMEOUT). In either case, the
20 service provider is responsible for proper cleanup of any processing that may have been performed in regards to this request.

The parameters to the WFPOpen request are stored within the Open Request. Those stored in the SpiRequest and SpiAsyncRequest objects are private and can only be accessed using a function. Fig. 9 is a table which
25 illustrates an example of each WFPOpen parameter and how it is accessed for an embodiment of the present invention.

For immediate processing, the method is:

HRESULT <specificSp>OpenRequest:: spImmediateProcessing(void).

This method, if implemented, is invoked by the service provider framework after all WFOpen request parameters have been verified. This includes version negotiation as described in the WOSA XFS API/SPI Programmer's Reference Revision 2.00 section 5.7. Service providers may not need to implement this method.

For deferred processing, the method is:

Boolean <specificSp>OpenRequest:: spDeferredProcessing(void).

This method allows the service provider to perform the actual processing associated with "opening a device". This may include performing PDH commands to initialize the device used by this service provider. This method is invoked every time WFOpen is called by the XFS Manager. The service provider must use the OpenRequest::openFailed() method if for some reason it determines that the open operation has failed. The openFailed() method allows the service provider framework to remove the session from its tables.

The parameters to the WFPRegister request are stored within the Register Request object. Those stored in the SpiRequest and SpiAsyncRequest objects are private and can only be accessed using a function. Fig. 10 is a table which illustrates an example of each WFPRegister parameter and how it is accessed for an embodiment of the present invention.

The parameters to the WFPSetTraceLevel request are stored within the Set Trace Level Request object. Those stored in the SpiRequest object are private and can only be accessed using a function. Fig. 11 is a table which illustrates an example of each WFPSetTraceLevel parameter and how it is accessed for an embodiment of the present invention.

Event processing is described in Section 3.11 in the WOSA XFS API/SPI Programmer's Reference Revision 2.00. The postEvent() method in the XfsEvent object is used to post event messages for WFP request complete and for the EXECUTE, SERVICE, USER, and SYSTEM events. XfsPost() is described hereinafter.

WFP request complete events or completion events for short, are Windows messages that are used to indicate that a given asynchronous WFP request is through processing. The service provider is responsible for posting completion events for any WFP request in which the service provider has implemented a spDeferredProcessing method that overrides the same method in the base class for that WFP request. Typically the service provider implements a spDeferredProcessing method for WFPOpen, WFPExecute, WFPGetInfo, and WFPCLose requests. The service provider framework posts all other completion events.

Each service provider is responsible for posting all Execute, Service, and User events described in the Programmer's Reference for their device. The service provider is also responsible for posting the System events described in the Device Status Changes section and the Hardware and Software Errors section of the Programmer's Reference. The service provider framework is responsible for posting the system events related to undeliverable messages and version negotiation errors.

For postEvent(), the method is:

```
void XfsEvent::postEvent(
    DWORD notificationMessage,
    XfsServiceProvider *serviceProvider,
    LPWFSRESULT lpWfsResult)
```

This method is used by service providers to send events and request completion notifications.

Parameters include notificationMessage which can take on the following values:

```
WFS_SERVICE_EVENT
WFS_USER_EVENT
WFS_SYSTEM_EVENT
WFS_EXECUTE_EVENT
```

Command Completion Messages (see WOSA XFS API/SPI Programmer's
Reference Revision 2.00 section 9.1.1)

Parameters also include:

5 serviceProvider is a handle unique to each XFS service provider. This handle is
provided to each service provider in the serviceProvider attribute in every Open
Request object.

hResult is the value placed in the hResult member of the WFSRESULT structure
pointed to by lpWfsResult

10 lpWfsResult is a pointer to a WFSRESULT structure. The service provider
framework allocates a WFSRESULT structure anytime a request object is
instantiated. The attribute name inside the request object is wfsResult. So, if the
event to be posted is a Request Complete event, the WFSRESULT structure from
the request object can be used; but only once. Any subsequent event postings for
that request must use the XfsEvent::allocateWfsResult() to allocate a new
15 WFSRESULT structure. If for any event, the WFSRESULT structure points to
additional data, XfsEvent::allocateMore() (see below) must be used to allocate
additional space.

For the SERVICE_EVENTS, USER_EVENTS, and SYSTEM_EVENTS,
postEvent sends event notification to every hWnd for this service that has
20 registered for the specified event. For EXECUTE_EVENTS, only the hWnd
associated with the hService for the execute request currently in progress receives
the event notification. The service provider is responsible for setting the hResult
and dwEventID member of the WFSRESULT structure. The RequestID,
hService, and tsTimestamp members of the WFSRESULT structure are set by the
25 postEvent method.

For allocateWfsResult, the method is:

HRESULT XfsEvent::allocateWfsResult(LPVOID * lppvData).

This method is used to allocate an XFS WFSRESULT structure.

Parameters include:

lppvData which is the address of the variable in which allocatWfsResult will place the pointer to the allocated memory.

If the function return is not WFS_SUCCESS, the return value is one of the following error conditions:

- 5 WFS_ERR_INVALID_POINTER if lppvData does not point to accessible memory.

WFS_ERR_OUT_OF_MEMORY if there is not enough memory available to satisfy to request. When calling WfMAllocateBuffer, the service provider framework uses both the WFS_MEM_SHARE and WFS_MEM_ZEROINIT

10 flags.

For allocateMore, the method is:

HRESULT XfxEvent::allocateMore(

ULONG ulSize

LPVOID lpvOriginal

- 15 LPVOID * lppvData)

This method is used to allocate a memory buffer, linking it to an existing one.

Parameters include:

ulSize which is the size (in bytes) of the memory to be allocated.

lpvOriginal which is the address of the original buffer to which the newly

- 20 allocated buffer should be linked.

lppvData which is the address of the variable in which allocatWfsResult will place the pointer to the allocated memory.

If the function return is not WFS_SUCCESS, the return value is one of the following error conditions:

- 25 WFS_ERR_INVALID_POINTER if a pointer parameter does not point to accessible memory.

WFS_ERR_OUT_OF_MEMORY if there is not enough memory available to satisfy to request.

WFS_ERR_INVALID_ADDRESS the lpvOriginal parameter does not point to a previously allocated buffer.

When calling WFMAllocateBuffer, the service provider framework uses both the WFS_MEM_SHARE and WFS_MEM_ZEROINIT flags.

- 5 As already mentioned, in an embodiment of the present invention, each service provider is required to instantiate a specific instance of its service provider object. That object is derived from the service provider framework's XFS Service Provider class as shown in Fig. 2. The XFS Service Provider class has virtual methods, such as forExecuteRequest, that will instantiate the request objects if not overridden by the same function in the service provider's derived class. So, for those WFP requests, such as WFPExecute, that require service provider processing, the service provider must override the request object instantiation method in the XFS Service Provider base class. Using the depositor service provider as an example, the method in the derived service provider class is, for example:
- 10

```
DepositorExecuteRequest *  
DepositorServiceProvider::forExecuteRequest( void )  
{
```

```
    DepositorExecuteRequest * request;  
20    request = new DepositorExecuteRequest;  
    return( request );
```

1. result buffer how do applications access.
2. cancel/abort should be the service providers responsibility to terminate the thread in a timely fashion.

- 25 Various preferred embodiments of the invention have been described in fulfillment of the various objects of the invention. It should be recognized that these embodiments are merely illustrative of the principles of the invention. Numerous modifications and adaptations thereof will be readily apparent to those

skilled in the art without departing from the spirit and scope of the present invention.

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2